Tutorial: Detect and track objects in real-time with OpenCV

OpenCV's Cascade Classifier is a machine learning framework for object detection. You can leverage a Cascade Classifier with a pretrained model to detect and track objects in real-time. A pretrained model can help you get started with your object detection program:

- To detect an object of interest, researchers train classifiers on large datasets over long periods of time. When this process is complete, the classifier generates a pretrained model.
- The pretrained model contains the features needed to detect an object.

What you will learn

- The fundamentals of object detection as they apply to OpenCV.
- How to create an object detection program with OpenCV and Python.

Before you begin

- Install OpenCV and Python. For more information, view Get Started.
- Ensure that you have the pretrained models, or `Cascade XML` files in your **OpenCV directory**:
    - Go to your **OpenCV directory** > Select the **data** folder > Select the **haarcascades** folder.

      The `haarcascades` folder contains `Haar-Cascade XML` files. These files are pretrained classifiers for different objects. For example, the `haarcascade_eye.xml` file is a classifier trained to detect eyes.

  If you do not have the Haar Cascade files, you must install the `openCV-contrib` module. To install this module, run the command:

  pip install opencv-contrib-python

- Prepare an input video or image if you do not plan to use an integrated camera, like a web camera. In this tutorial, the input video is highway surveillance footage. To download the input video:
    - Go to `Program Files` > Download the `samplevideo.mp4` file.
- Prepare or download a pretrained model or `Haar-Cascade XML` file. The classifier you choose depends on your application.

  To follow along with this tutorial for car detection download `cars.xml`, the `Haar-Cascade XML` file for car detection:

    - `Go to Program Files > Download the cars.xml file.`
    - `Place the cars.xml file in your data folder.`

Steps

1. Read, display and write a video with the OpenCV HighGUI API.

2. Apply image preprocessing techniques
3. Initialize the Cascade Classifier

Read, display and write a video with the OpenCV HighGUI API.

Read, display, and write videos within OpenCV's HighGUI. Load local videos or capture streams from integrated cameras, like your web camera, interact with video frames in real-time, and write the video to your files.

1. Read a video or capture an integrated camera stream, like a web cam:
    a. Import `cv2`
    b. Create an instance of the `VideoCapture()` class:
        - To read a video file, pass your video file as an argument:

          `cv2.VideoCapture('your video path.mp4')`
        - To read a video from a web camera, pass `0` as an argument.
    c. **Optional.** Set the resolution of your video frames. You might want to find the size of your input video frames to configure the size of your output frames.

       In the following code snippet, the first parameter passed to the `get()` function is the property constant `CAP_PROP_FRAME_WIDTH`. This parameter returns the width of your frames. For more properties accesible through this method, view Enumerations.

       `frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))`

Try it out

```
import cv2

# Open your input video file
cap = cv2.VideoCapture('your_video_path.mp4')

# Get the frame width and height
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

print(f"Frame Width: {frame_width}, Frame Height: {frame_height}")

# Release the video capture
cap.release()
```

2. Write the frames to an output file and display the video:

   a. Create an instance of the `VideoWriter()` class.

   videoWriter()

Parameter	Description
File path	Set the path or filename to the output video file.
Fourcc	To compress and save the video frames, provide a FourCC codec. A codec is software that compresses or decompresses a video. For example, you can use the codec 'm', 'p', '4', 'v' for MP4 video compression.
Frame rate	The frame rate is the number of frames per second (fps). Frame rates depend on your computational resources and object detection application. For example, video surveillance systems typically use a frame rate of 30 fps. To process a 3D model in real-time, you might require a frame rate up to 100 fps.
Resolution	Set the resolution of the video frames.

   The following code snippet shows how to create an instance of the `VideoWriter()` class.

```
file_path = 'your output video file path.mp4'
fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
fps = 30.0
resolution = (1280, 1720)
```

   b. Capture and process the input video frames. In the following code sample:

   - The loop reads frames from the input video source, and each iteration of the loop reads the next frame. The `read()` function reads frames until the end of the video is read.
   - The `frame` variable stores each frame. `frame` is an OpenCV variable that stores the image data of each frame that is reached.

```
# Loop through the video frames

while True:

      ret, frame = cap.read()

 # Check if the end of the video has been reached
 if not ret:
     break
```

c. Display your video in a HighGUI window with the `imshow()` function. The following code snippets shows how to call the `imshow()` function to display your video frames.

```
# Show the video in a window named "frame"

# Store each frame that is shown with the frame variable

cv2.imshow("frame", frame)
```

d. Write the frames to the output video file with the `write()` function:
`out.write(frame)`

e. To keep the GUI window open until a key press, and process your video frames, call the `waitKey()` function. When you do not use this function, the GUI window will open and close almost immediately, and you will not be able to view or store the frames.

You can use this function anywhere in your code where you want the GUI to wait for input.

waitKey()

Parameters	Description
delay	The amount of time in milliseconds that the GUI window will stay open for until a key press. To keep the window open until a key press, pass `0` as the argument

The following code snippets show two different methods to use the `waitKey()`function.

```
# Close the window after 5000 milliseconds or a key press
cv2.waitKey(5000)
```

```
# Waits 25 milliseconds for a key press. If the key press is `q`,
the loop breaks and the program ends.
if cv2.waitKey(25) & 0xFF == ord('q'):
break
```

e. Close the video capture object.

```
cap.release()
```

d. **Optional.** To destroy the GUI window, call the `destroyAllWindows()` function. Typically, the GUI window closes when your program ends. However, if you run your program from the Python terminal, it might stay open until you close the terminal.

Try it out

```
import cv2

cap = cv2.VideoCapture('test-video.mp4')

while True:
    ret, frame = cap.read()

    if not ret:
        break

cv2.imshow('Video', frame)

# Add a twenty-five millisecond delay and wait for a key press
# Break the loop if the 'q' key is pressed
if cv2.waitKey(25) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

Apply image preprocessing techniques

Certain image properties, such as noise, color, and contrast, change the efficacy of the object detection framework. When you remove noise and color from your input video, the results of your object detection framework might improve. You can change the look of your video with OpenCV's filters.

The following techniques are recommendations. For all OpenCV image preprocessing functions, view Image Filtering.

Reduce noise with a Gaussian Blur

A common method to remove noise from an image is to blur the image. However, excessive noise reduction can remove fine details and edges from objects. Your object detector needs these details to detect the objects effectively.

When you want to reduce noise in an image, but maintain the fine details, you can use a gausian blur. The GaussianBlur() function uses Gaussian distribution to distribute a blur effect to each pixel and it's neighborhood pixels. This function adapts the blur level to the neighborhood pixel values.

gausianBlur()

Parameters	Description
src	Set your input video or image file path.
kernel size	The blur intensity depends on the kernel size; the more significant the kernel size, the more intense the blur effect will be. For more detailed information on kernels, view the following section.
Standard Deviation (Sigma)	Set the standard deviation to `0` to automatically generate the strength of the blur based on your kernel size. For more information on standard deviation in this function, view Standard deviation in GausianBlur().

The following code snippet shows how to use the `gausianBlur()` function.

```
Cv2.GaussianBlur(frame, ksize, std)
```

Kernel size

Kernel size is fundamental topic in computer vision. A kernel is a small filter, like a window, that moves across each pixel in an image. The kernel size controls the extent of a filter, such as a blur.

Suppose you want to blur your video or image. The extent of the blur on central pixel, or the current pixel that the kernel is on, depends on it's neighborhood pixels. In a 3 x 3 kernel, the neighborhood would be the eight pixels around the central pixel:

1 2 3

4 5 6

7 8 9

All pixels in a kernel have a value based on their grayscale or color intensity. Suppose you want to apply a blur to your video with the GausianBlur() function. The goal of this filter is to blur the central pixel with its neighbors, as a result:

- **A small kernel size (neighborhood), like a 3 x 3, creates a minor blur effect.** In a kernel, the goal is to average the color or grayscale intensities of the central pixel with those of its neighbors. The extent of blur applied to the central pixel is determined by the weighted average of the central pixel and its neighboring pixels.

  The greater the number of neighbors within a kernel, the stronger the weighted average effect and the more intense the blur. For example, in a 3 x 3 kernel, the central pixel is averaged with its eight neighboring pixels. The smaller neighborhood size results in a less pronounced blur effect.

- **A large kernel size, like a 5 x 5, creates a more significant blur effect.** In a 5 x 5 kernel, the goal is to average the central pixel with its twenty-four neighborhood pixels. The larger weighted average creates a more noticeable blur.

Standard deviation in the GausianBlur() function

Together, the kernel size and standard deviation control the extent and intensity of the blur effect. The `GaussianBlur()` function uses a standard deviation to control the distribution, or intensity, of the blur.

- A high standard deviation leads to a stronger blur of the central pixel with its neighborhood.
- A lower standard deviation results in a milder blur of the central pixel and its neighbors.

If you set the standard deviation to `0`, the correct blur level will be set automatically based on the kernel size.

Dilate each frame

The `dilate()` function expands the size bright regions in an image, or image regions in the foreground. You might use this function to make objects more prominent in an image. For example, if a car's edges blends in with it's shadows, you can apply the `dilate()` function to make the vehicle edges more distinct. The `dilate()` function only changes the size and shape of image regions.

dilate()

Parameters	Description
src	Set your input video or image file path.
kernel size	The dilation intensity depends on the kernel size; the more significant the kernel size, the more intense the dilate will be. For more detailed information on kernels, view Kernels.
Number of iterations	Set the number of iterations for the dilate operation.

Convert the frames to grayscale

The Cascade Classifier uses a pretrained model that was trained on grayscale images. To help detect objects better, you can convert your frames to grayscale with the `cvtColor()` function. However, your application might need color. For example, you might want to create a program that detects all red cars on a highway.

cvtColor()

Parameters	Description
src	Set your input video or image file path.
Color space conversion method	For color to gray conversion, use the flag BGR2GRAY. For more information on color conversion codes, see Changing Colorspaces

Apply laplacian edge detection

Edge detection can improve the result of your object detection algorithm by convert each frame to binary. Edge detection is different from grayscale, as a binary image has only two pixel colors, typically black and white.

This function highlights areas in a frame where the pixel intensity changes rapidly. To apply laplacian edge detection to your video or image, call the `laplacian()` function.

laplacian()

Parameters	Description
src	Set your input video file path.
ddepth	Decide how intense the detected edges will be. In the following code sample, the output data type cv2,CV_8U is used. For more information on this data type, view One Important Matter
ksize	Set the kernel size. For more information, view Kernels.

Try it out

```
import cv2
import datetime

# Replace 'your_video_path' with the path to your video file or the sample
file
video_path = 'your_video_path.mp4'

cap = cv2.VideoCapture(video_path)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Define the output video file path
output_video_path = f'output_{datetime.date.today()}.mp4'

# Write the processed frames to an output file path
out = cv2.VideoWriter(output_video_path, cv2.VideoWriter_fourcc('m', 'p',
'4', 'v'), 10, (frame_width, frame_height))

while True:
    # Capture each frame of the video
    ret, frame = cap.read()
    if ret:
        # Apply image processing (e.g., Gaussian blur, Laplacian, etc.) here
        frame = cv2.GaussianBlur(frame, (5, 5), 0)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = cv2.Dilate(frame, (5,5), 2)

        # Display the processed frame
        cv2.imshow("frame", frame)
        out.write(frame)

        # Break the loop with the 'q' key
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Initialize the Cascade Classifier

A Cascade Classifier is a machine learning framework commonly used for object detection. This framework uses trained classifiers to find patterns in an image that are consistent with the object you want to detect.

Haar-like features

Suppose you want to train a cascade classifier to detect cars. Thousands of images with cars (positive images), and images without cars (negative images), are tested by classifier. These series of tests collect Haar-like features from the positive images.

Haar-like features are universal patterns that you find in an object, like wheels on a car. These features are rectangular patterns of pixel values.

In this tutorial, you will use a pretrained classifier that has previously created Haar-like features for an object.

Load the Pretrained Cascade Classifier

To initalize the Cascade Classifier:

1. Call the CascadeClassifier() function and pass a Haar cascades file as a parameter. In the following code snippet, a Haar Cascades for cars is combined with the CascadeClassifier().

   ```
   cv2.CascadeClassifier('cars.xml')
   ```

   a. Call the `detectMultiScale()` function. This function looks for objects at different sizes and locations. Next, the function returns a list of `(x,y,w,h)` rectangle coordinates around each object.

   detectMultiScale()

Required parameters	Description
image	Set your input video or image file path.
scaleFactor	A small scale factor, for example 1.02, will slow down the object detection rate, but improve the chance of detection of the object of interest at all scales. A large scale factor, for example 2, will result faster detection, but objects on a smaller scale or in the distance might be missed.

Required parameters	Description
minNeighbors	When you set a high value for this parameter, like 3, your system will look for multiple similar detections grouped before it decides a region is your object of interest. In some applications, like face detection on a crowded street, a high value could result in false negatives because the system waits for groups of detections before identification. A low value, like1, looks for less evidence of the object around the region. You might get more false positives with a low value if there are objects that could resemble the object of interest around the area.

In the following code snippet, a moderate `scaleFactor` value and a low `minNeighbors` value are set as parameters for `detectMultiScale()` function.

A low value for the `scaleFactor` makes the algorithm more likely to detect objects on a smaller scale or, cars in the distance. In the input video, a highway surivellance video, there are often cars that overlap. With a low `minNeighbors` value, the system will be more sensitive to individual instances of the cars, and more likely to detect overlapping cars.

```
import cv2

# Load the pre-trained car detection classifier car_cascade =
cv2.CascadeClassifier('cars.xml')

# Load the input image or video frame image = cv2.imread('sample-
image.jpg') # For image, use 'sample-image.jpg'; for video,
capture frames from 'sample-video.mp4'

# Detect cars and return locations with detectMultiScale() cars =
car_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=1, minSize=(30, 30))
```

b. Draw a bounding box around each object with the `rectangle()` function. In object detection, a bounding box is a rectangular region that surrounds the object of interest.

rectangle()

Parameters	Description
src	The amount of time in milliseconds that the GUI window will stay open for until a key press. To keep the window open until a key press, pass `0` as the argument
(x, y)	Top-left corner of rectangle.
(x+w, y+h)	Bottom-right corner of rectangle. (x, y) are the coordinates of the top-left corner. w and h are the width and height.
Rectangle color	Choose a BGR color. For example, (0, 0, 255) for a red rectangle.
Rectangle line width	Set the width of the rectangle border.

In the following code snippet, the `rectangle()` function prints rectangles with the (x, y, w, h) coordinates from the previous step.

```
for (x, y, w, h) in cars: cv2.rectangle(frame, (x,y), (x+w,y+h), (0,0,255), 2)
```

Try it out

```
import cv2
import datetime

# Replace 'your_video_path' with the path to your video file or the sample
file
video_path = 'your_video_path.mp4'

cap = cv2.VideoCapture(video_path)
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Define the output video file path
output_video_path = f'output_{datetime.date.today()}.mp4'

# Write the processed frames to an output file path
out = cv2.VideoWriter(output_video_path, cv2.VideoWriter_fourcc('m', 'p',
'4', 'v'), 10, (frame_width, frame_height))

while True:
    # Capture each frame of the video
    ret, frame = cap.read()
    if ret:
        # Apply image processing (e.g., Gaussian blur, Laplacian, etc.) here
        frame = cv2.GaussianBlur(frame, (5, 5), 0)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = cv2.Laplacian(src=frame, ddepth=cv2.CV_8U, ksize=3)

        # Load the car detection classifier (replace 'your_classifier.xml'
with the actual XML file)
        car_cascade = cv2.CascadeClassifier('your_classifier.xml')

        # Detect objects of interest in each frame
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=1)

        # Draw rectangles around detected objects
        for (x, y, w, h) in cars:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

        # Display the processed frame
        cv2.imshow("frame", frame)
        out.write(frame)

        # Break the loop with the 'q' key
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
```

```
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```